

```

1  '''
2  Question 1: Using Pandas Perform Merge and Join.
3  '''
4  import pandas as pd
5
6  df1 = pd.DataFrame({
7      'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
8      'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
9
10 df2 = pd.DataFrame({
11     'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
12     'salary': [70000, 80000, 120000, 90000]})
13
14 print("--- DataFrame 1 (df1) ---")
15 print(df1)
16
17 print("\n--- DataFrame 2 (df2) ---")
18 print(df2)
19
20 print("\n=====")
21 print("Merge using left_on='employee' and right_on='name'")
22 merged_on_columns = pd.merge(df1, df2, left_on="employee",
23     right_on="name").drop('name', axis=1)
24 print(merged_on_columns)
25
26 # Merge using 'left_index' and 'right_index'
27 df3 = pd.DataFrame({
28     'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
29     'hire_date': [2004, 2008, 2012, 2014]
30 })
31
32 df1a = df1.set_index('employee')
33 df2a = df3.set_index('employee')
34
35 print("\n=====")
36 print("--- (df1a) ---")
37 print(df1a)
38
39 print("\n--- (df2a) ---")
40 print(df2a)
41
42 print("Merge using left_index=True and right_index=True")
43 merged_on_index = pd.merge(df1a, df2a, left_index=True, right_index=True)
44 print(merged_on_index)
45
46 '''
47 Output:
48 --- DataFrame 1 (df1) ---
49     employee      group
50 0      Bob  Accounting
51 1      Jake  Engineering
52 2      Lisa  Engineering

```

```

53 3      Sue      HR
54
55 --- DataFrame 2 (df2) ---
56   name  salary
57 0   Bob   70000
58 1  Jake   80000
59 2  Lisa  120000
60 3   Sue   90000
61
62
63 Merge using left_on='employee' and right_on='name'
64   employee      group  salary
65 0      Bob  Accounting   70000
66 1     Jake  Engineering   80000
67 2     Lisa  Engineering  120000
68 3      Sue      HR      90000
69
70
71 --- (df1a) ---
72           group
73 employee
74 Bob      Accounting
75 Jake      Engineering
76 Lisa      Engineering
77 Sue      HR
78
79 --- (df2a) ---
80   hire_date
81 employee
82 Lisa      2004
83 Bob       2008
84 Jake      2012
85 Sue       2014
86 Merge using left_index=True and right_index=True
87   group  hire_date
88 employee
89 Bob      Accounting      2008
90 Jake      Engineering      2012
91 Lisa      Engineering      2004
92 Sue      HR      2014
93
94 '''
95
96
97
98 '''
99 2nd Question: Using Pandas Perform Concat and Append.
100 '''
101 import pandas as pd
102 data_x = {
103     'A': ['A0', 'A1'],
104     'B': ['B0', 'B1']
105 }
106 x = pd.DataFrame(data_x, index=[0, 1])
107 data_y = {

```

```

108     'A': ['A2', 'A3'],
109     'B': ['B2', 'B3']
110 }
111 y = pd.DataFrame(data_y, index=[2, 3])
112
113 # Assign x's index to y's index (Duplicate Indices)
114 y.index = x.index
115
116 print("--- Original DataFrames ---")
117 print("x:")
118 print(x)
119 print("\ny (now has duplicate indices [0, 1]):")
120 print(y)
121
122 print("\n" + "="*40)
123 print("--- Result of pd.concat([x, y]) ---")
124 print(pd.concat([x, y]))
125
126 print("\n" + "="*40)
127 print("--- Result of pd.concat([x, y], ignore_index=True) ---")
128 print(pd.concat([x, y], ignore_index=True))
129
130 '''
131 Output:
132 --- Original DataFrames ---
133 x:
134     A  B
135 0  A0  B0
136 1  A1  B1
137
138 y (now has duplicate indices [0, 1]):
139     A  B
140 0  A2  B2
141 1  A3  B3
142
143 =====
144 --- Result of pd.concat([x, y]) ---
145     A  B
146 0  A0  B0
147 1  A1  B1
148 0  A2  B2
149 1  A3  B3
150
151 =====
152 --- Result of pd.concat([x, y], ignore_index=True) ---
153     A  B
154 0  A0  B0
155 1  A1  B1
156 2  A2  B2
157 3  A3  B3
158 '''
159
160 '''
161 Question 3: Demonstrate NumPy Structured arrays
162 '''

```

```

163 import numpy as np
164
165 # --- 1. Creating a Structured Array ---
166 print("--- 1. Creating a Structured Array ---")
167 data_type = [
168     ('ID', 'i4'),
169     ('Product', 'U10'),
170     ('Price_USD', 'f8')
171 ]
172
173 inventory = np.array([
174     (101, 'Laptop', 1200.50),
175     (102, 'Monitor', 350.00),
176     (103, 'Keyboard', 75.99)
177 ], dtype=data_type)
178
179 print("Structured Array (Inventory Records):\n", inventory)
180 print("\nDType Structure:\n", inventory.dtype)
181 print("-" * 20)
182
183
184 print("\n--- 2. Accessing Structured Array Data ---")
185 print("Accessing the 'Product' field (all product names):")
186 products = inventory['Product']
187 print(products)
188 print(f"Result Type: {type(products)}, DType: {products.dtype}")
189 print("-" * 50)
190
191 print("Accessing the record at index 1 (Monitor):")
192 record_1 = inventory[1]
193 print(record_1)
194 print(f"Result Type: {type(record_1)}, DType: {record_1.dtype}")
195 print("-" * 50)
196
197 print("Accessing 'Price_USD' from the record at index 0 (Laptop's Price):")
198 price = inventory[0]['Price_USD']
199 print(price)
200
201 '''
202 Output:
203 --- 1. Creating a Structured Array ---
204 Structured Array (Inventory Records):
205 [(101, 'Laptop', 1200.5 ) (102, 'Monitor',  350.  )
206  (103, 'Keyboard',    75.99)]
207
208 DType Structure:
209 [('ID', '<i4'), ('Product', '<U10'), ('Price_USD', '<f8')]
210
211
212 --- 2. Accessing Structured Array Data ---
213 Accessing the 'Product' field (all product names):
214 ['Laptop' 'Monitor' 'Keyboard']
215 Result Type: <class 'numpy.ndarray'>, DType: <U10
216
217 Accessing the record at index 1 (Monitor):

```

```
218 (102, 'Monitor', 350.0)
219 Result Type: <class 'numpy.void'>, DType: [('ID', '<i4')], ('Product',
220 '<U10')], ('Price_USD', '<f8')]]
221
222 Accessing 'Price_USD' from the record at index 0 (Laptop's Price):
223 1200.5
224
225
226 '''
227 Question 4: Demonstrate 3 different examples of Broadcasting
228 '''
229 import numpy as np
230
231 print("--- Example 1: ---")
232 M = np.ones((2, 3))
233 a = np.arange(3)
234 print("M + a: ")
235 print(M + a)
236
237
238 print("--- Example 2: ---")
239 a = np.arange(3).reshape((3, 1))
240 b = np.arange(3)
241 print("a + b: ")
242 print(a + b)
243
244 print("--- Example 3: ---")
245 M = np.ones((3, 2))
246 a = np.arange(3)
247 print("M + a[:, np.newaxis] =")
248 print(M + a[:, np.newaxis])
249
250 '''
251 Output:
252 --- Example 1: ---
253 M + a:
254 [[1. 2. 3.]
255  [1. 2. 3.]]
256 --- Example 2: ---
257 a + b:
258 [[0 1 2]
259  [1 2 3]
260  [2 3 4]]
261 --- Example 3: ---
262 M + a[:, np.newaxis] =
263 [[1. 1.]
264  [2. 2.]
265  [3. 3.]]
266 '''
267
268 '''
269 Question 5: On a given dataset perform Aggregations
270 '''
271 import numpy as np
```

```
272
273 data = np.array([5, 12, 18, 12, 5, 20, 25, 18, 12])
274
275 aggregations = {
276     "1. Minimum (Min)": np.min(data),
277     "2. Maximum (Max)": np.max(data),
278     "3. Mean (Average)": np.mean(data),
279     "4. Median (Middle Value)": np.median(data),
280     "5. Standard Deviation (Std)": np.std(data),
281     "6. Variance (Var)": np.var(data)
282 }
283
284 print(f"Dataset Array: {data}")
285 print("-" * 40)
286 print("Statistical Aggregations (NumPy Only):")
287 print("-" * 40)
288
289 for name, value in aggregations.items():
290     print(f"{name}: {value:.2f}")
291
292 '''
293 Output:
294 Dataset Array: [ 5 12 18 12  5 20 25 18 12]
295
296 Statistical Aggregations (NumPy Only):
297
298 1. Minimum (Min): 5.00
299 2. Maximum (Max): 25.00
300 3. Mean (Average): 14.11
301 4. Median (Middle Value): 12.00
302 5. Standard Deviation (Std): 6.35
303 6. Variance (Var): 40.32
304 '''
305
306 '''
307 Question 6: Demonstrate Universal Functions
308 '''
309 import numpy as np
310
311 # Set print options for clean, concise output
312 np.set_printoptions(precision=4, suppress=True)
313
314 print("="*20)
315 print("NumPy Universal Functions (Ufuncs) Demonstration")
316 print("="*20)
317
318 X = np.arange(5)
319 Y = np.arange(5, 10)
320 A = np.array([-2.5, 0.0, 1.0, np.pi / 2, -1.0])
321
322 print(f"Input Array X: {X}")
323 print(f"Input Array Y: {Y}")
324 print(f"Input Array A: {A}\n")
325
326 print("--- 1. Basic Arithmetic (Binary) ---")
```

```

327 print(f"X + Y (np.add): {X + Y}")
328 print(f"Y - X (np.subtract): {Y - X}")
329 print(f"X * Y (np.multiply): {X * Y}")
330 print(f"Y / (X + 1) (np.divide): {np.divide(Y, X + 1)}")
331 print(f"Y % 3 (np.remainder): {Y % 3}\n")
332
333 print("--- 2. Absolute Value (Unary) ---")
334 absolute_A = np.abs(A)
335 print(f"np.abs(A):\n{absolute_A}\n")
336
337 print("--- 3. Trigonometric Functions (Unary) ---")
338 sine_A = np.sin(A)
339 print(f"np.sin(A): {sine_A}")
340
341 cosine_A = np.cos(A)
342 print(f"np.cos(A): {cosine_A}")
343
344 tangent_A = np.tan(A)
345 print(f"np.tan(A): {tangent_A}")
346
347 A_clipped = np.clip(A, -1, 1) # Ensure values are within [-1, 1] range
348 arcsin_A = np.arcsin(A_clipped)
349 print(f"np.arcsin(A_clipped): {arcsin_A}\n")
350
351 print("--- 4. Exponents and Logarithms (Unary) ---")
352 exp_X = np.exp(X)
353 print(f"np.exp(X) (e^x):\n{exp_X}")
354
355 log2_Y = np.log2(Y)
356 print(f"np.log2(Y):\n{log2_Y}")
357
358 log_Y = np.log(Y)
359 print(f"np.log(Y):\n{log_Y}")
360
361 print(f"np.expm1(X * 0.0001):\n{np.expm1(X * 0.0001)}")
362
363 '''
364 Output:
365 =====
366 NumPy Universal Functions (Ufuncs) Demonstration
367 =====
368 Input Array X: [0 1 2 3 4]
369 Input Array Y: [5 6 7 8 9]
370 Input Array A: [-2.5      0.      1.      1.5708 -1.      ]
371
372 --- 1. Basic Arithmetic (Binary) ---
373 X + Y (np.add): [ 5  7  9 11 13]
374 Y - X (np.subtract): [5 5 5 5 5]
375 X * Y (np.multiply): [ 0  6 14 24 36]
376 Y / (X + 1) (np.divide): [5.      3.      2.3333 2.      1.8      ]
377 Y % 3 (np.remainder): [2 0 1 2 0]
378
379 --- 2. Absolute Value (Unary) ---
380 np.abs(A):
381 [2.5      0.      1.      1.5708 1.      ]

```

```
382
383 --- 3. Trigonometric Functions (Unary) ---
384 np.sin(A): [-0.5985  0.      0.8415  1.      -0.8415]
385 np.cos(A): [-0.8011  1.      0.5403  0.      0.5403]
386 np.tan(A): [ 7.4702e-01  0.0000e+00  1.5574e+00  1.6331e+16 -1.5574e+00]
387 np.arcsin(A_clipped): [-1.5708  0.      1.5708  1.5708 -1.5708]
388
389 --- 4. Exponents and Logarithms (Unary) ---
390 np.exp(X) (e^x):
391 [ 1.      2.7183  7.3891 20.0855 54.5982]
392 np.log2(Y):
393 [2.3219 2.585  2.8074 3.      3.1699]
394 np.log(Y):
395 [1.6094 1.7918 1.9459 2.0794 2.1972]
396 np.expm1(X * 0.0001):
397 [0.      0.0001 0.0002 0.0003 0.0004]
398 '''
399
400 '''
401 Question 7: Demonstrate Operating on Null Values
402 '''
403 import pandas as pd
404 import numpy as np
405
406 print("="*30)
407 print("Scenario 1: Dropping Columns that are COMPLETELY Null")
408 print("="*30)
409
410 data_drop = {
411     'ID': [101, 102, 103],
412     'Value': [45, 67, 88],
413     'Notes': [np.nan, np.nan, np.nan],
414     'Comments': ['Good', None, 'Great']
415 }
416 df_drop = pd.DataFrame(data_drop)
417
418 print("--- Initial DataFrame (df_drop) ---")
419 print(df_drop)
420
421 null_col_check = df_drop.isnull().all()
422 print("\n--- Check: df_drop.isnull().all() ---")
423 print(null_col_check)
424
425 df_dropped = df_drop.dropna(axis=1, how='all')
426
427 print("\n--- Result: DataFrame after dropping ALL-null columns ---")
428 print(df_dropped)
429
430 print("="*30)
431 print("Scenario 2: Filling Partial Null Values with the Scalar 0")
432 print("="*30)
433
434 df_fill = df_dropped.copy()
435
436 print("--- DataFrame before filling (df_fill) ---")
```



```

437 print(df_fill)
438
439 print("\n--- Null value count per column before filling ---")
440 print(df_fill.isnull().sum())
441
442 scalar_value = 0
443 df_filled = df_fill.fillna(value=scalar_value)
444
445 print(f"\n--- Result: DataFrame after filling nulls with '{scalar_value}' ---")
446 print(df_filled)
447
448
449 '''
450 Output:
451 =====
452 Scenario 1: Dropping Columns that are COMPLETELY Null
453 =====
454 --- Initial DataFrame (df_drop) ---
455      ID  Value  Notes  Comments
456 0  101     45   NaN     Good
457 1  102     67   NaN     None
458 2  103     88   NaN     Great
459
460 --- Check: df_drop.isnull().all() ---
461 ID           False
462 Value        False
463 Notes         True
464 Comments    False
465 dtype: bool
466
467 --- Result: DataFrame after dropping ALL-null columns ---
468      ID  Value  Comments
469 0  101     45     Good
470 1  102     67     None
471 2  103     88     Great
472
473 Scenario 2: Filling Partial Null Values with the Scalar 0
474 =====
475 --- DataFrame before filling (df_fill) ---
476      ID  Value  Comments
477 0  101     45     Good
478 1  102     67     None
479 2  103     88     Great
480
481 --- Null value count per column before filling ---
482 ID           0
483 Value        0
484 Comments     1
485 dtype: int64
486
487 --- Result: DataFrame after filling nulls with '0' ---
488      ID  Value  Comments
489 0  101     45     Good
490 1  102     67         0

```

```
491 2 103 88 Great
492
493
494 '''
```